Project Document

Team 2 - Anno Domini

Version 1.4

[Date: 03-29-2019] CPSC 4900 Spring 2019

Sequoia Kanies – Project Manager/Scrum Master

Bowen Wexler – Development Manager

Brandon Hough – System Analyst Manager

Jonathan Mensi – Quality Assurance Manager

Austin Lynn – Application Architect

SECTION 1 - INTRODUCTION

With this project we will be working with small groups of programmers and project managers to implement a software project for playing various solitaire games. The class as a whole will meet and organize the various games and put them together into one large project. This process will be facilitated by the application architects from each team, as they work to make each individual project work together on one platform and Graphical User Interface (GUI). The main menu, therefore, will be a class-wide assignment, while each individual game will be built by each team respectively. Our team is working to create the game *Anno Domini*. The user of the software will interact with the main menu to select which game they desire to play, then will be directed by this main menu to one of the various other games. The main menu will be customizable by the player so that only the favorite games will be visible and they will have the ability to mute the subtle background music.

1.1 PURPOSE

The purpose of this assignment is to teach students how to build large software systems with a team of developers, architects and managers. The reason that large projects are so important to have experience with, is that, for business applications in the real world, large systems are the primary projects encountered. We will thus, by the end of this class, be able to work in conjunction with local or remote groups of developers, managers, users or business associates.

1.2 SCOPE

The first instance of the importance of scope for this project is in relation to the duties of each team. The teams will need to first organize as a class so that the projects will work in conjunction. Each team's application architect and development manager will be responsible for working to build a basic template for the creation of the individual projects. Then, once the main menu is created, the individual teams will work to create their solitaire game, to be accessible through a link from the main menu. Limitations of the project are best described by the roles of each team above but also include the necessity for simplicity. If each team were to work alone and create a stand-alone system that was complex, then try to put them together, there would be problems. These problems would include the refactoring of code to make them fit together into one larger project and reorganization of the specifications and language or operating system requirements. Thus, we must, as a class, work to create one project using Java and therefore facilitate the use of multiple operating systems. Some other limitations would be that we are coding the game in the programming language Java as well as having a deadline of only 6 weeks.

1.3 TERMINOLOGY

Operating System (OS) - The software that supports a computer's basic functions, such a scheduling tasks, executing applications and controlling peripherals (dictionary.com).

Programming Language - a vocabulary or syntax set used for computers to perform specific tasks.

Graphical User Interface (GUI) - a visual way of interacting with a computer using items such as windows, icons and menus used by most modern operating systems (dictionary.com).

Object Oriented Design (OOD) - Programming that uses classes of code with attributes. Each object has a set of attributes and methods that can be used on the object.

Architecture - The design and models, including UML diagrams, that describe how the system will be organized and built. This does not include the actual programming of the system.

Implementation - The actualization of the architecture, which includes the programs and related documents such as commenting.

User Requirements - Statements, in natural language plus diagrams, of what services of the system is expected to provide to system users and the constraints under which it must operate (Sommerville, pg. 102).

System Requirements - Detailed descriptions of the software systems functions, services and operational constraints written for the developers of the system (Sommerville, pg. 55).

Stakeholder - A person that is affected by the system in some way and anyone who has a legitimate interest in it and often have conflicting requirements and needs (Sommerville, pg. 103).

Functional Requirements - Statements of services the system should provide, how the system should react to particular inputs, and how the system should behave in particular situations. May also explicitly state what they system should not do (Sommerville, pg. 105).

Non-functional Requirements - Constraints on the services or functions offered by the system which include timing constraints, constraints on the development process, and constraints imposed by standards (Sommerville, pg. 105).

Frames Per Second (FPS) - A measurement of how many frames will be displayed on a monitor every second.

1.4 REFERENCES

This section will be worked on iteratively as the project progresses.

- *Software Engineering*, Ian Sommerville, 10th Edition, Publisher: Person
- https://www.dictionary.com/

SECTION 2 - MANAGEMENT

2.1 GENERAL DESCRIPTION

- The ***Project Manager/Scrum Master*** is Sequoia Kanies and his role is to lead in developing and maintaining the project schedule, organize and leading the Team Meetings/Scrum sessions, oversee compilation and delivery of the project document, and reporting project status to Upper Management.
- The ***Application Architect*** is Austin Lynn and his role is to have an overall vision for the solution/design, lead in developing and maintaining the System Architecture Specification, and coordinate the traceability of design and development (RTM).
- The ***Development Manager*** is Bowen Wexler and his role is to lead in developing and maintaining System Design Specifications, coordination of developer resources, and code production and delivery.
- The ***Quality Assurance Manager*** is Jonathan Mensi and his role is to lead in developing and maintaining the Test Plan and Test Specifications, coordinating test activities, and validation of developed software.
- The ***System Analyst Manager*** is Brandon Hough and his role is to organize and lead requirements elicitation activities, lead in developing and maintaining the Requirements Specification, and coordinate the traceability of requirements and design (RTM).

2.2 SOFTWARE VALIDATION

Our team plans to use a GitHub repository to make necessary changes to fit the card game we were given, *Anno Domini*. We will then ensure that each of the four games will be able to run on the common GUI framework with the other 3 games created by the other teams in the class. We will then test and validate our code with different use cases conducted by the QA Manager, Jonathan Mensi.

2.3 CONFIGURATION MANAGEMENT PLAN

With the aid of the Scrum Masters and Application Architects for each team, the configuration management will be handled via the project plan. This will be dedicated to be documented by the Scrum Master, and the configuration of the main menu will be handled by the various Architects and Developers. By sprint two, the main menu has been created and each team is working on their individual games. The application architects are working to keep the versions lined up and have created a file share, to facilitate this teamwork. The source code documentation will be included with this sprint submission. There are not a wide range of versions of this project, but instead are updated versions of the original. Each of the changes made will be tracked on Github if we need to go back to a previous version of the code. That means that instead of maintaining a large bank of older versions, the useful and current version is worked on. The reason that this is possible, is that this is not an enterprise system, with years of different versions. There is one, large team working on the project as a whole, and this allows for the thorough understanding of the project by each. There are not different teams, working at different times, on different versions, but rather a large team working together. This is broken down into smaller groups, lead by the architects of the main platform. Thus, the data sheets, source code, and in turn, various versions are systematically melded into a final product. This means that the necessity for configuration management is minimized, as the project has a very limited number of versions to consider. The primary diversity in style or code would be limited to the individual teams, who will report on the source code of their pieces via sections IV and V of the project document below.

2.4 PROJECT SCHEDULE

The project plan will detail the schedule to be followed and will be updated along with the requirements, as both change and evolve. For sprint 1, the schedule is broken down into weeks, roles, and the percentage complete that each task is. For specifics, check the project plan. Sprint 2 is made up of the completion of the main menu, the start on the individual code segments that represent each game, the finalization of the UML diagrams to model the systems, and the updating of the requirements and documentation. Sprint 2 will be used to create the game objects such as cards, stacks, and the game-specific organizations of cards. These will be finished in sprint 3, but the general layout of the entities and the UI will be implemented in Sprint 2. This will allow for sections IV and V to be worked on to document the code. Sprint 3 will consist of finishing the individual games and making sure that the data is sent from the game to the main menu, to keep track of statistics. This consists, more specifically, of having the specified game attributes which will be visible from the main menu, having data transferable between the two, being able to select the favorite games of the user, sound settings, and other functional specifications. The nonfunctional requirements will begin to be realized at this point. Once the games are functional, the ways in which the user interacts with the game will be modified at the GUI level, to make it more appealing. The individual games will have submenus to show the rules of the game you are playing. In short, the focus will go from focusing on the functionality of the games, to the documentation and testing of the system in sprint 4. This will include barraging the system with various anomalous use cases, repetitive and edge testing, and by trying to break the main menu or each game, to determine the shortcomings therein. At this point, the documentation can be finished by completing sections V and VI of this, the project document. All references and resources will be added to the documentation at this time as well.

SECTION 3 - REQUIREMENTS

3.1 BUSINESS REQUIREMENT SPECIFICATION (BRS)

3.1.1 SETUP

- The game shall draw cards at random to ensure fair gameplay for each player.
- The game shall let the user pick from one of the four games to play.
- The game shall allow the player to check which games are displayed on the main screen.

3.1.2 THE RULES

- The game shall follow the official rules of the the game *Anno Domini*.

3.1.3 THE EXPERIENCE

- The game shall provide an immersive experience that will relate closely to play a real game of cards.

3.1.4 STATISTICS

- The game shall display a menu screen that displays player statistics for each game.

3.1.5 SOUNDS

- The game shall have background music and sound effects to immerse the player into the game.
- The game shall have the ability to mute the background noise.

3.2 FUNCTIONAL REQUIREMENT SPECIFICATION (FRS)

3.2.1 SETUP

- The game shall have a standard setup when a new game begins.
- The game shall be able to share a common GUI framework in order to integrate the other three games as well. The other games are; *Argos*, *American Toad*, and *Aztec Pyramids.*

3.2.2 THE RULES

- The game shall follow the official rules of the the game *Anno Domini*.
- The game shall check the user moves validity and inform the user if an incorrect move was made.
- There shall be a *Rules* menu where users can check the rule set for the given game.

3.2.3 THE EXPERIENCE

- The game shall be easy to navigate through the menus and have smooth gameplay.

3.2.4 SYSTEM INPUTS

- The game shall take inputs from the mouse in order to select menus as well as click and drag cards around the GUI.

3.2.5 SYSTEM OUTPUTS

- The game shall implement a subtle background music to entice the user to play the game more.
- The game shall implement sound for when a user lays down a card and the connotation of the sound will be based on if the card can or can not be placed in that spot.
- There will be congratulations music when a user has won the game.

3.2.6 STATISTICS

- The game shall keep track of users specific statistics, including win ratio, win count, loss count, average time, and best time.
- The game shall save the scores of each user into a text document
- The game shall have a menu to display leaderboard scores

3.3 NONFUNCTIONAL REQUIREMENTS

3.3.1 PERFORMANCE CONSIDERATIONS

- There should not be any issues considering the technology and computing power of computers nowadays. The game will run somewhere between 30 and 60 FPS.

3.3.2 USER INTERFACE

- This will be presented in a non-cluttered GUI with professional graphics.

3.3.3 LANGUAGE

- The game shall be written in the programming language Java.
- The game shall implement a common GUI framework between the four teams in order to have all four games work in conjunction to become a suite.

3.3.4 OPERATING ENVIRONMENT

- The game shall be able to run on any desktop machine that has a Java environment. This would include Windows, Mac OS, and Linux.
- The game will not be compatible for iOS or Android apps during this iteration.

3.3.5 DEVELOPMENT STYLE

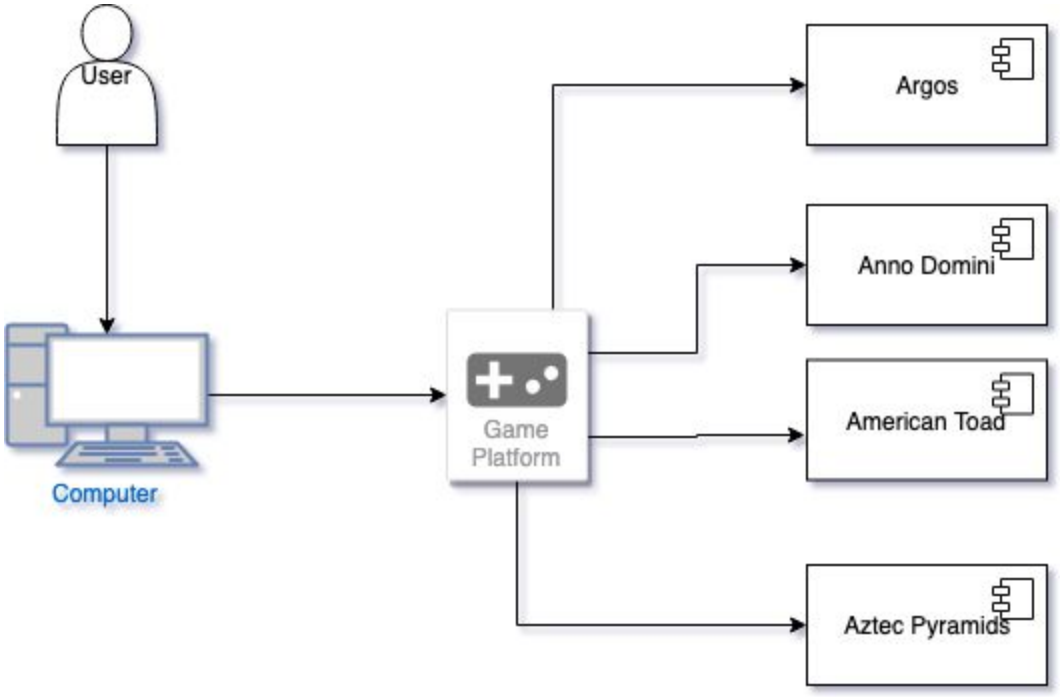
- This game shall be developed using the Scrum agile methods
- This game shall be developed by producing deliverables with each increment, or sprint.

SECTION 4 – DESIGN

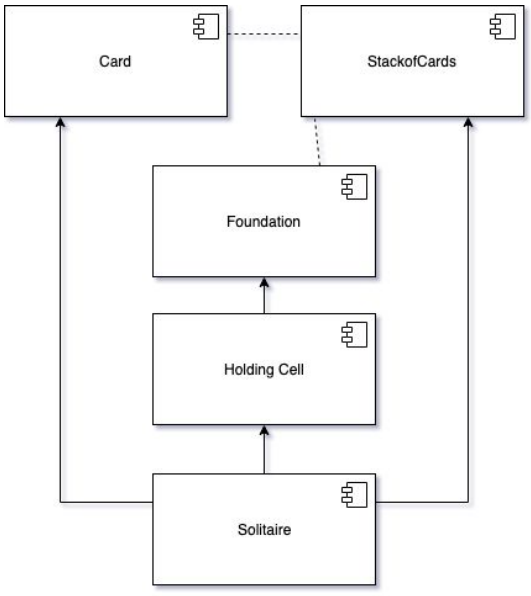
4.1 SYSTEMS ARCHITECTURE

The Anno Domini game is a Java local application that is to be run on a computer with Java SE-1.8.

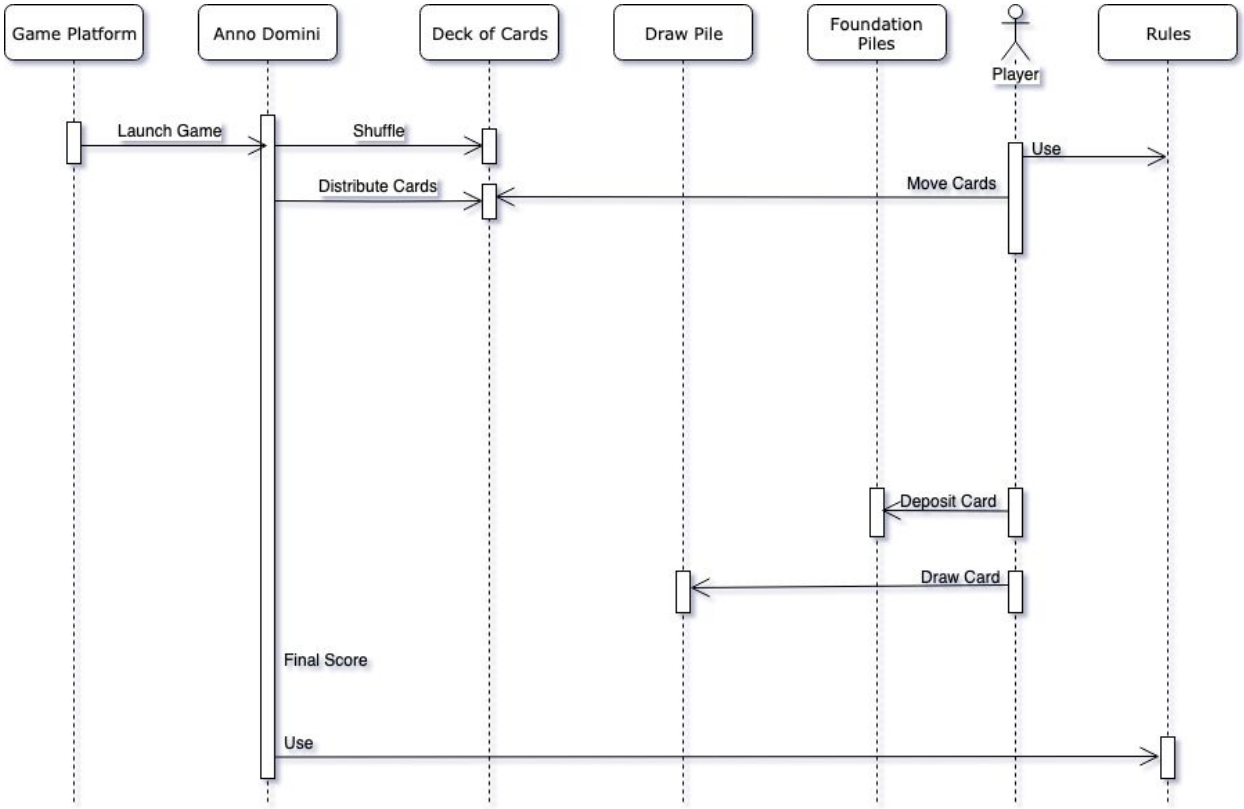
4.2 CONFIGURATION DESIGN



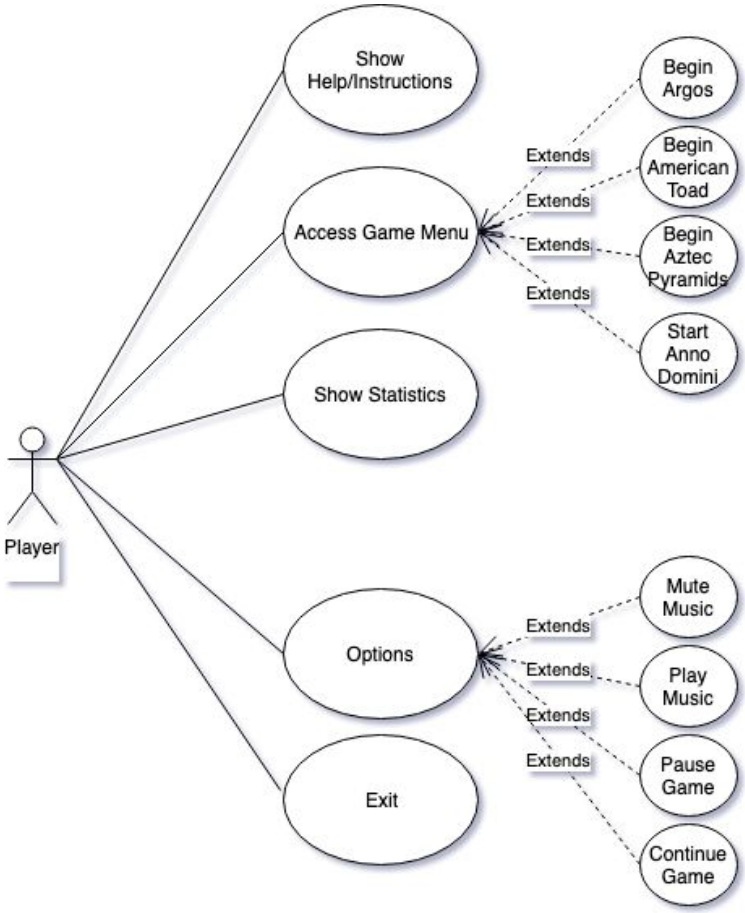
The user will use a computer that has Java capabilities to run Anno Domini. This is a high, abstract level diagram.



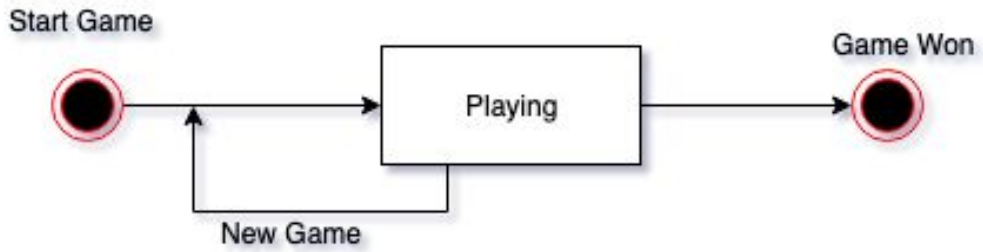
This is a component diagram illustrating the relations between the different Java components.



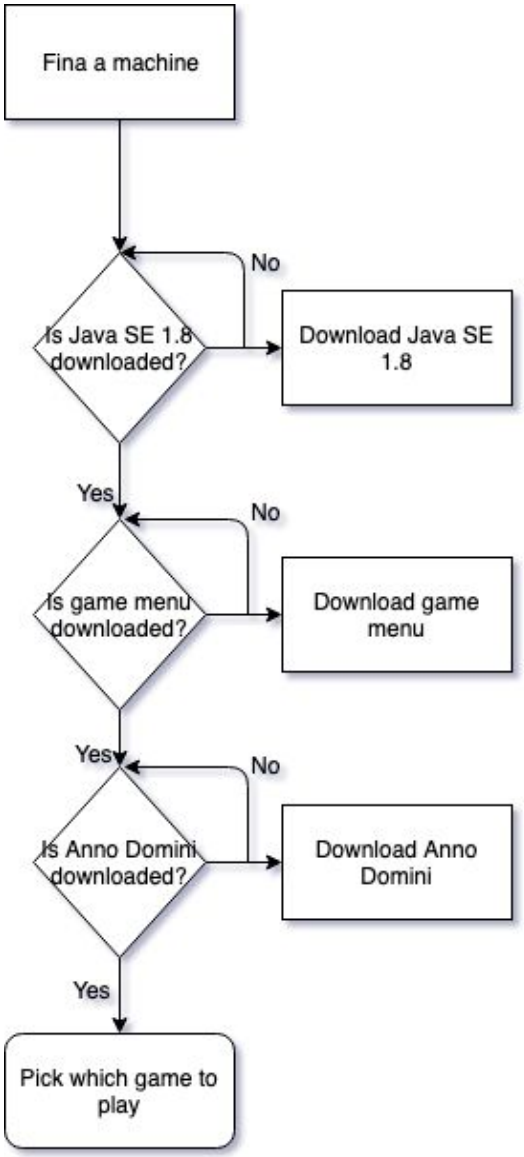
This is a sequence diagram illustrating the course of events that takes place over time when playing Anno Domini.



This is a use-case diagram illustrating the interactions between the user and the systems functionalities.

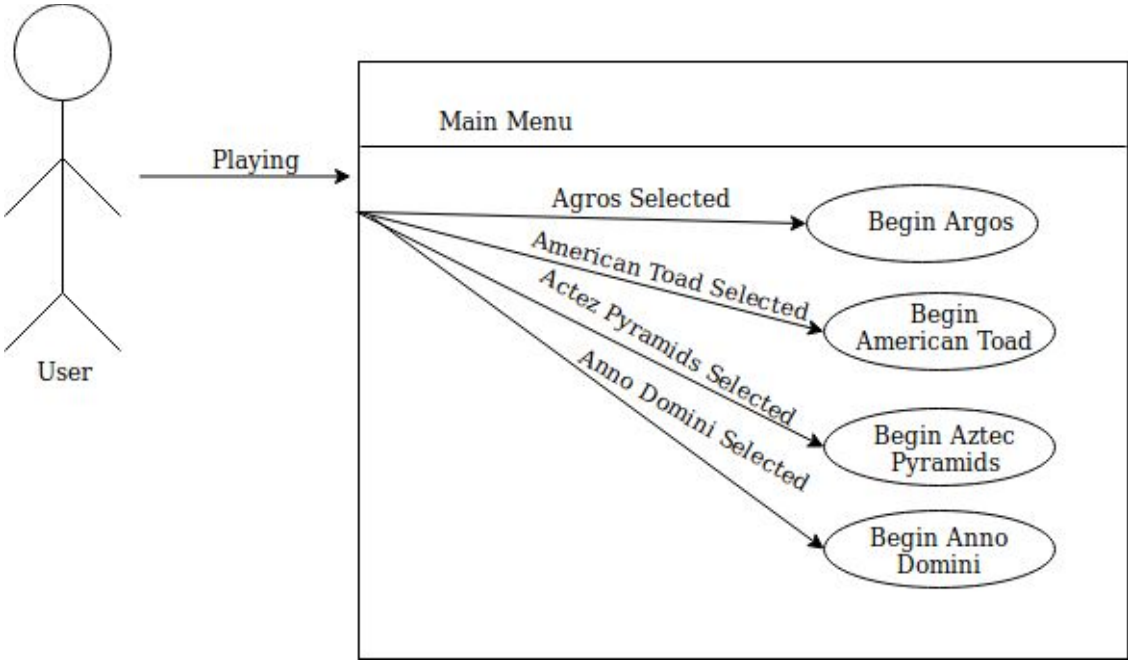


This is a state diagram showing the different states available while playing the game.

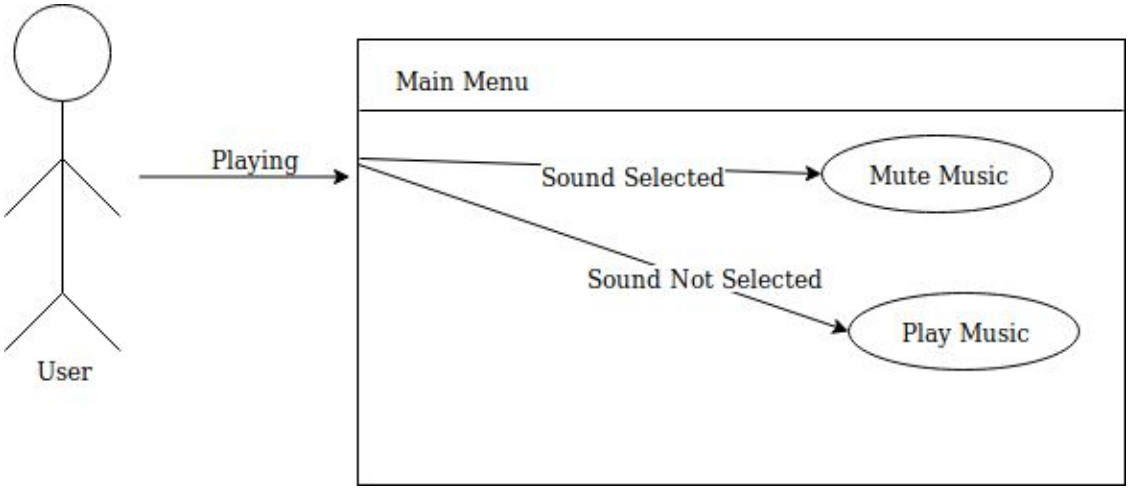


This is an activity diagram showing the activities taken in order to play this game.

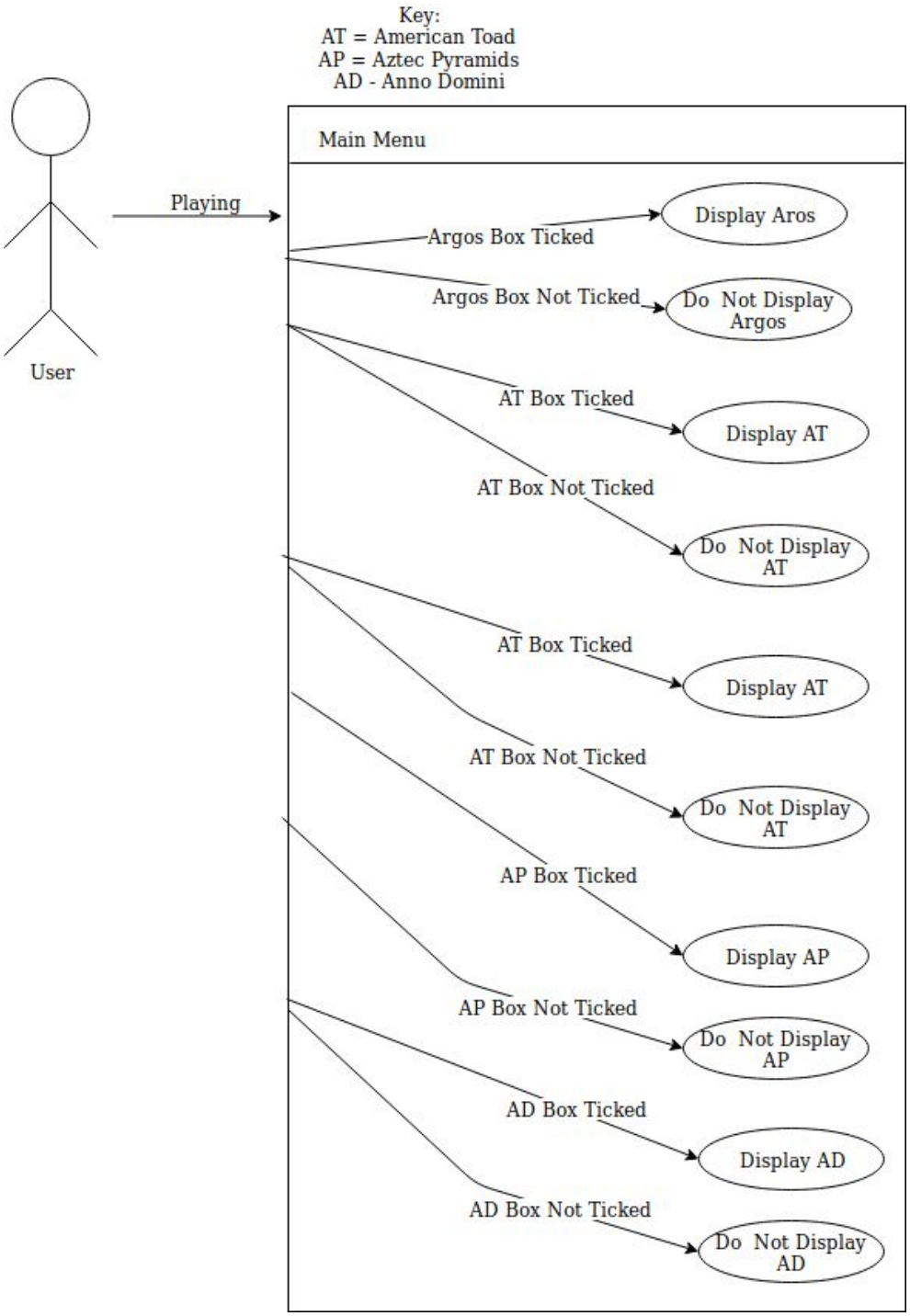
4.2 TEST CASE DIAGRAMS



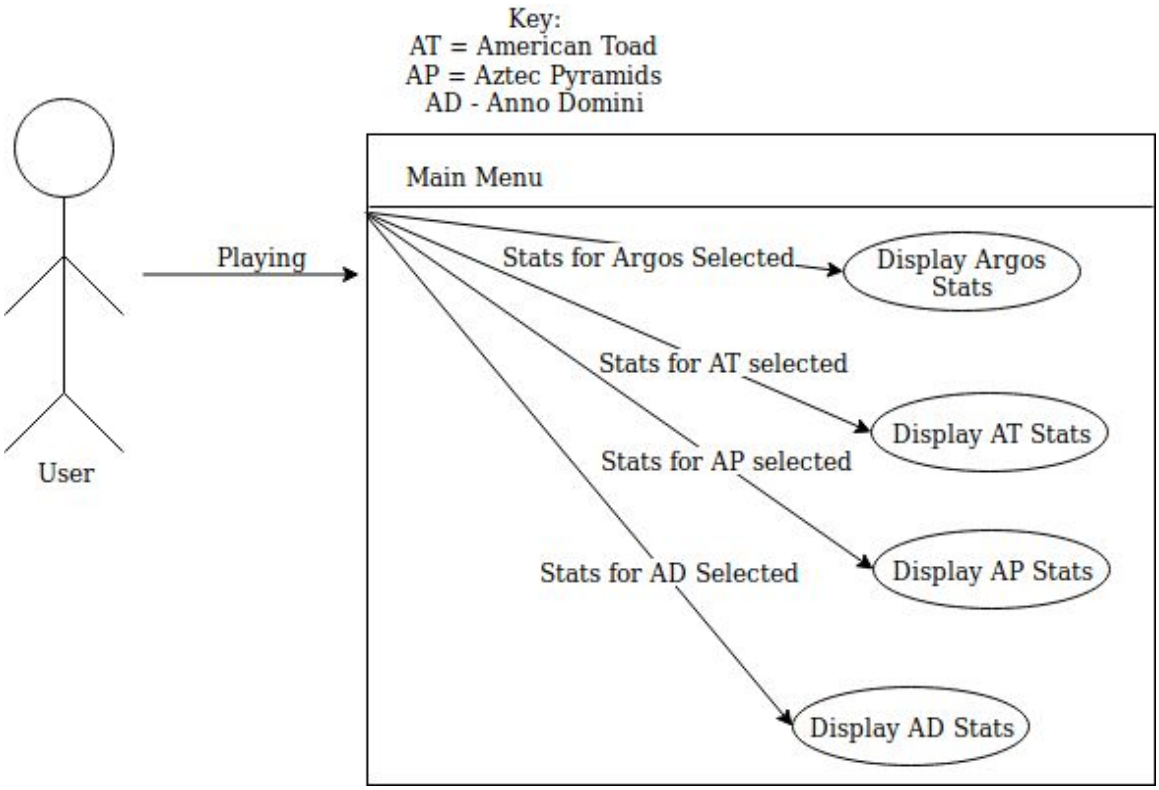
This is the test Case ID 01 which is defined in section *7.3 Test Cases*. If this is possible, then that test case will be validated.



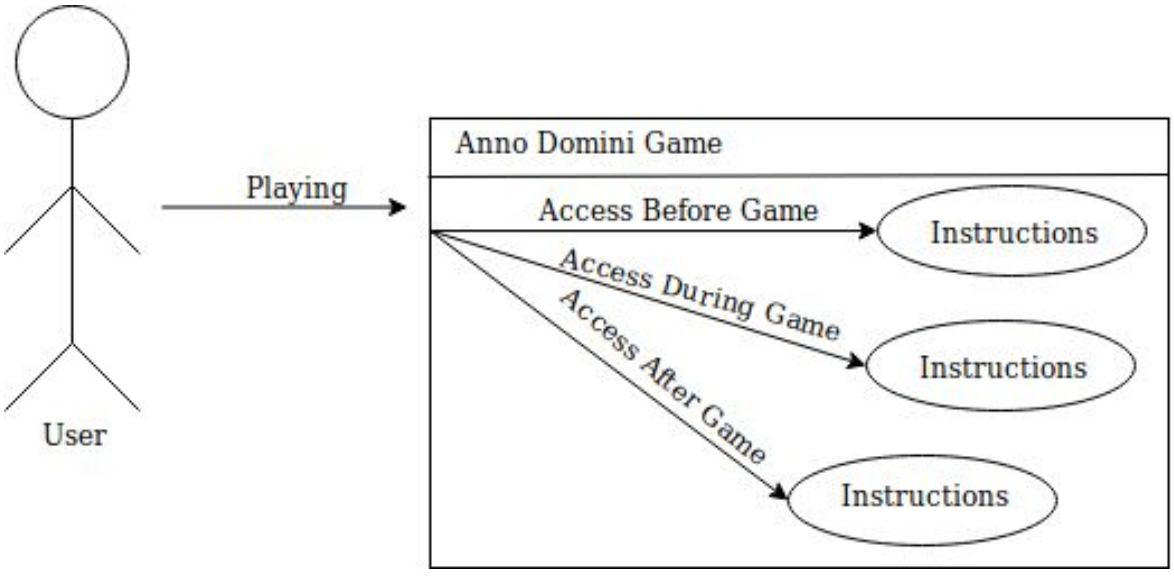
This is the test Case ID 02 which is defined in section *7.3 Test Cases*. If this is possible, then that test case will be validated.



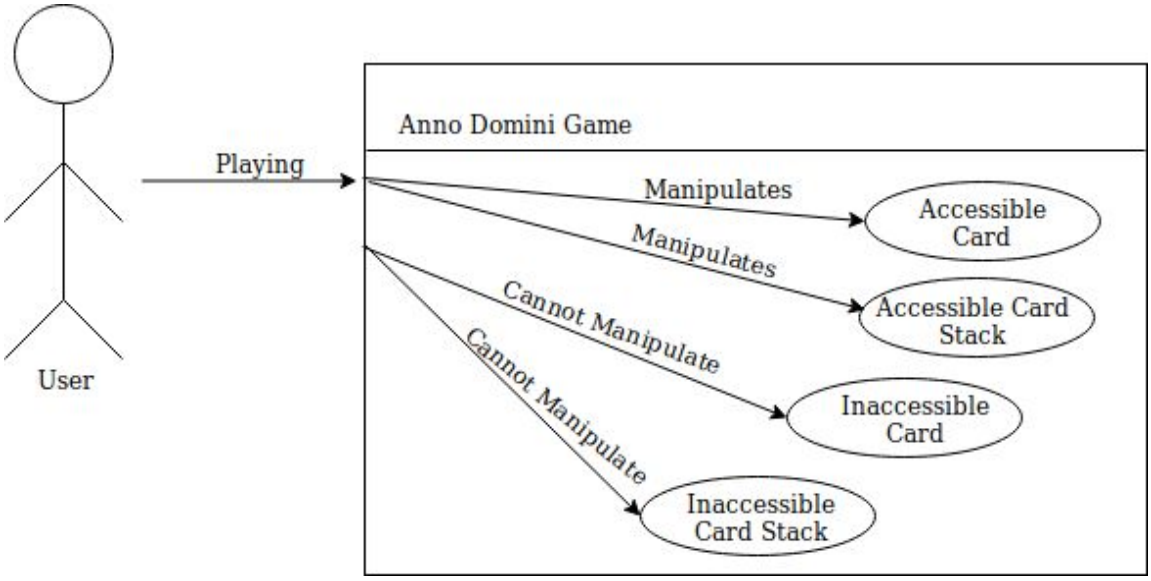
This is the test Case ID 03 which is defined in section *7.3 Test Cases*. If this is possible, then that test case will be validated.



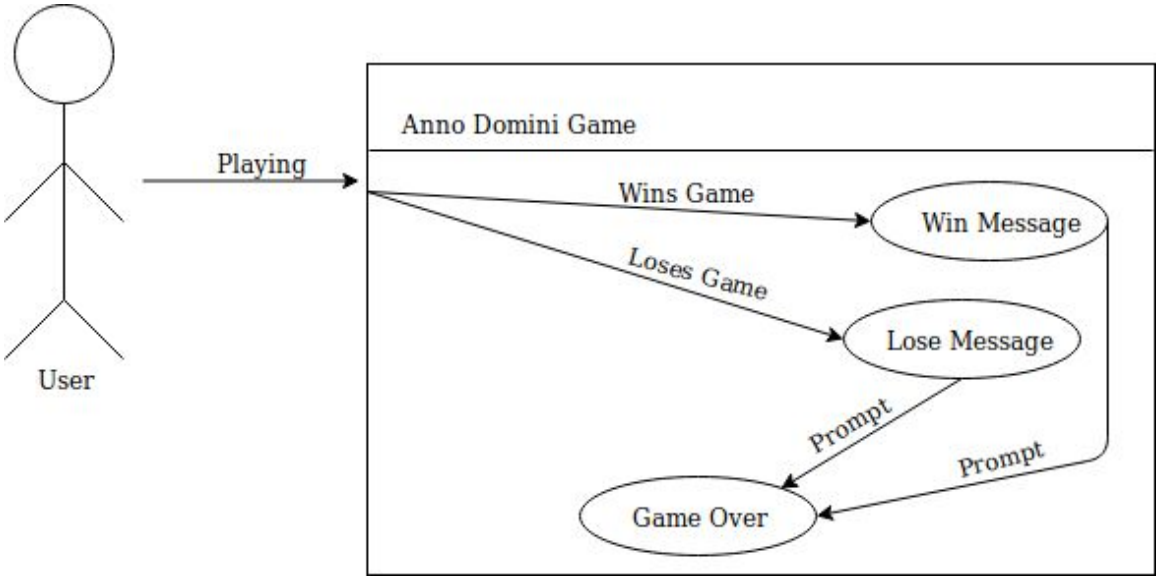
This is the test Case ID 04 which is defined in section *7.3 Test Cases*. If this is possible, then that test case will be validated.



This is the test Case ID 05 which is defined in section *7.3 Test Cases*. If this is possible, then that test case will be validated.



This is the test Case ID 06 which is defined in section *7.3 Test Cases*. If this holds true, then that test case will be validated.



This is the test Case ID 07 which is defined in section *7.3 Test Cases*. If this holds true, then that test case will be validated.

4.3 TECHNICAL DESIGN

The game is a Java version of Anno Domini. The users are specifically Mr. Hare's Software Engineering class of Spring 2019, but it is not limited to just class use. It can be used for anyone who has the source code (or JAR file) and a machine to run
it on.

SECTION 5 - IMPLEMENTATION STRATEGY

When approaching Implementation for our Project we kept in mind the three large topics relating to

implementation.

1. Reuse
2. Configuration Management
3. Host target development

Reuse- This was very important for us from a starting point as we decided as a class to reuse a base version of Solitaire already created using java by Warren Godone Maresca. The following link will take you to the initial version he created: https://github.com/warrengm/Solitaire

Reusing this software gave us a crucial starting point with some basic classes such as Card, StackOfCards, Suit, Stack, StackADT, and Queue. Following this Warren had already implemented a few different versions of solitaire games such as Spider, Yukon, Klondike, and base Solitaire. This approach on reusing software saved our programmers lots of up front time from creating base classes, and base games we can use. Using this software to our advantage let each group break down their focus onto their particular games while also adding in extra details on top. Software reuse in this manor is a big deal as a majority of what programmers do in the real world is reusing old software to create the new software.

Configuration management- As a class we understood going in that we are working on an entire project that the end result will need to come together into one large program. Similar to how different teams may focus on different parts of software development in the real world we have split apart our duties as a class while maintaining a comfortable place to track our progress, and ensure no versions of our project get lost no matter which development team is working on it. We have achieved this by doing two things so we can track each team / development.

1. Github usage - https://github.com/insomniac94/CPSC4900Repo - We created a github very early on that will host all of our versions / codes. It is very easy to track any changes a development manager may make to their individual codes which is then pushed to the entire project code.
2. Discord Server - Upfront we created a discord server and invited everyone in our class to join it. To briefly describe this its like a gigantic room that we can all discuss, communicate, and share what is happening within each group. It's a great place for us to communicate amongst ourselves about the project. Bowen our development manager exports all discord logs to our file exchange on blackboard so that you can follow the communication occurring.

Using these two methods at the start of our project enabled us as an entire class to focus on Configuration Management. Track the versions of the games we are created as well as provide an open communication platform for each of our development teams.

Lastly, Host-target development is a key focus when Implementing our project. A majority of our programmers are using Eclipse as an IDE they are familiar with to compile and execute their code. However our target machines are any machines that have Java installed on them and can run an executable.

Meeting the set out requirements was the next big step in approaching our Implementation.

SECTION 6 - USER DOCUMENTATION

6.1 INSTALLATION GUIDE

The first thing you will need to do to install Anno Domini on your machine would be to install the latest version of Java onto your machine. You can find java at the following link below:

Java: Here

After reaching the Java website go ahead and hit the big red “free java download” button in the middle of your screen. Next the Java website should detect whether you are using MacOS, Windows, or another operating system. After reaching this webpage go ahead and hit the “Agree and start free download” button. It may take some time to install all of Java onto your machine so be patient!

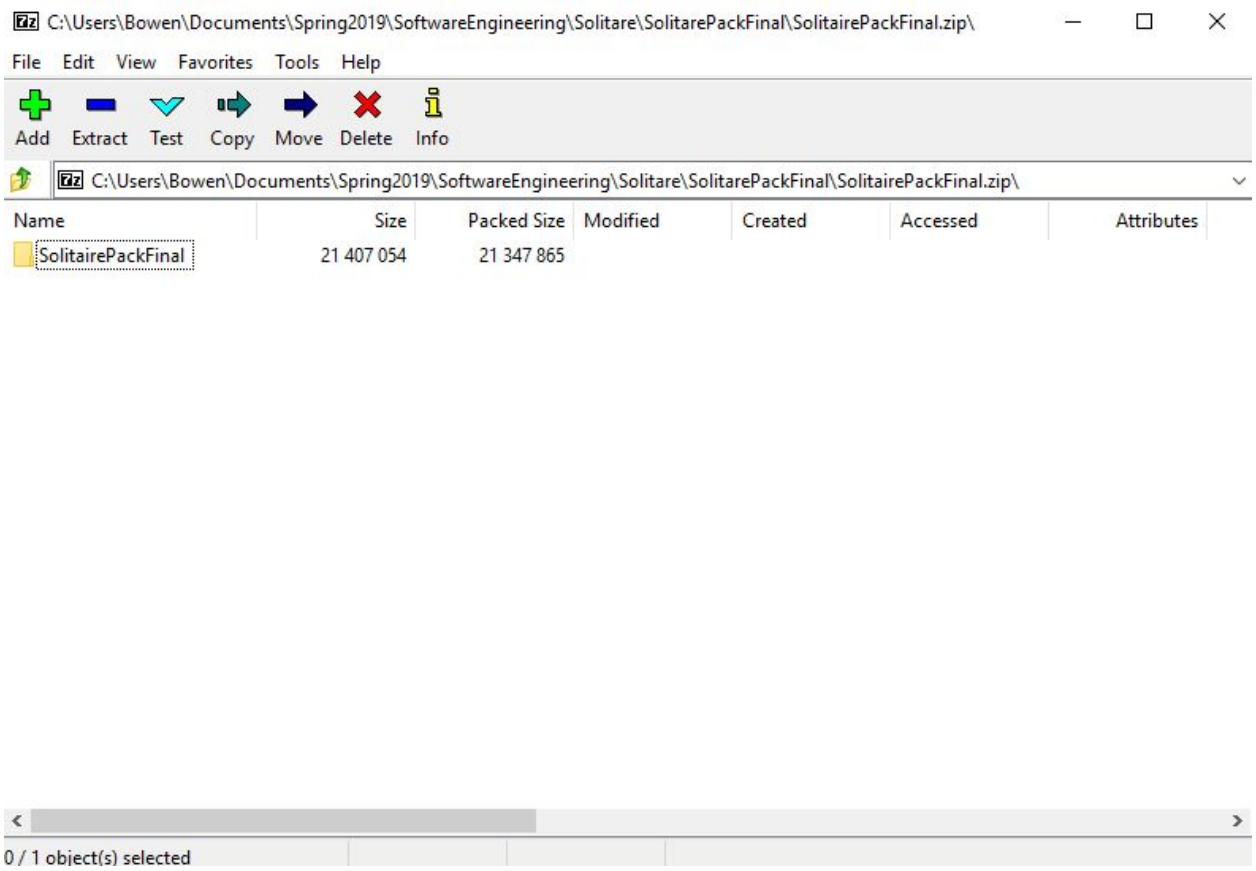
After installing Java onto the machine you are using the next thing you need to do is download the file to run our game! The link provided below will take you to a webpage where you can download our final Solitaire game that has Anno Domini included.

Solitaire: Here

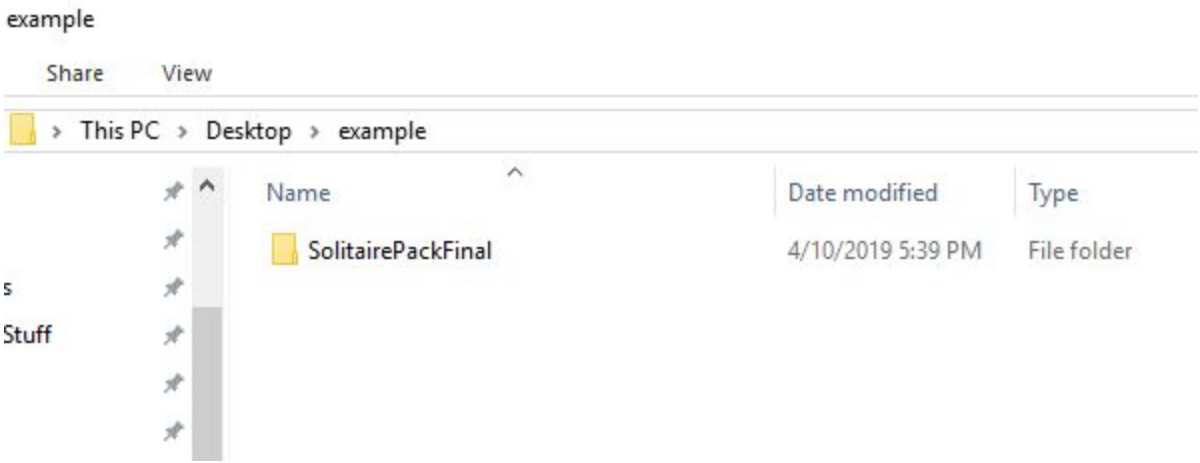
Upon reaching the Mediafire webpage you will want to click the big green download button that is provided. This should download a .zip file onto your computer. If you do not have a way to unzip this file already installed on your computer you can install 7zip at the following link below. (Note: Macs should have a way to zip/unzip built into their OS.)

7zip: Here

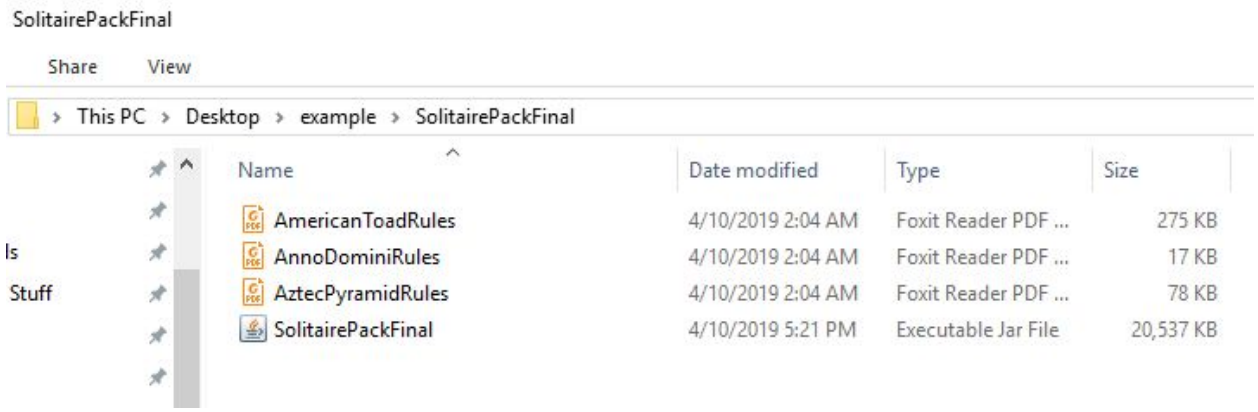
Before unzipping:



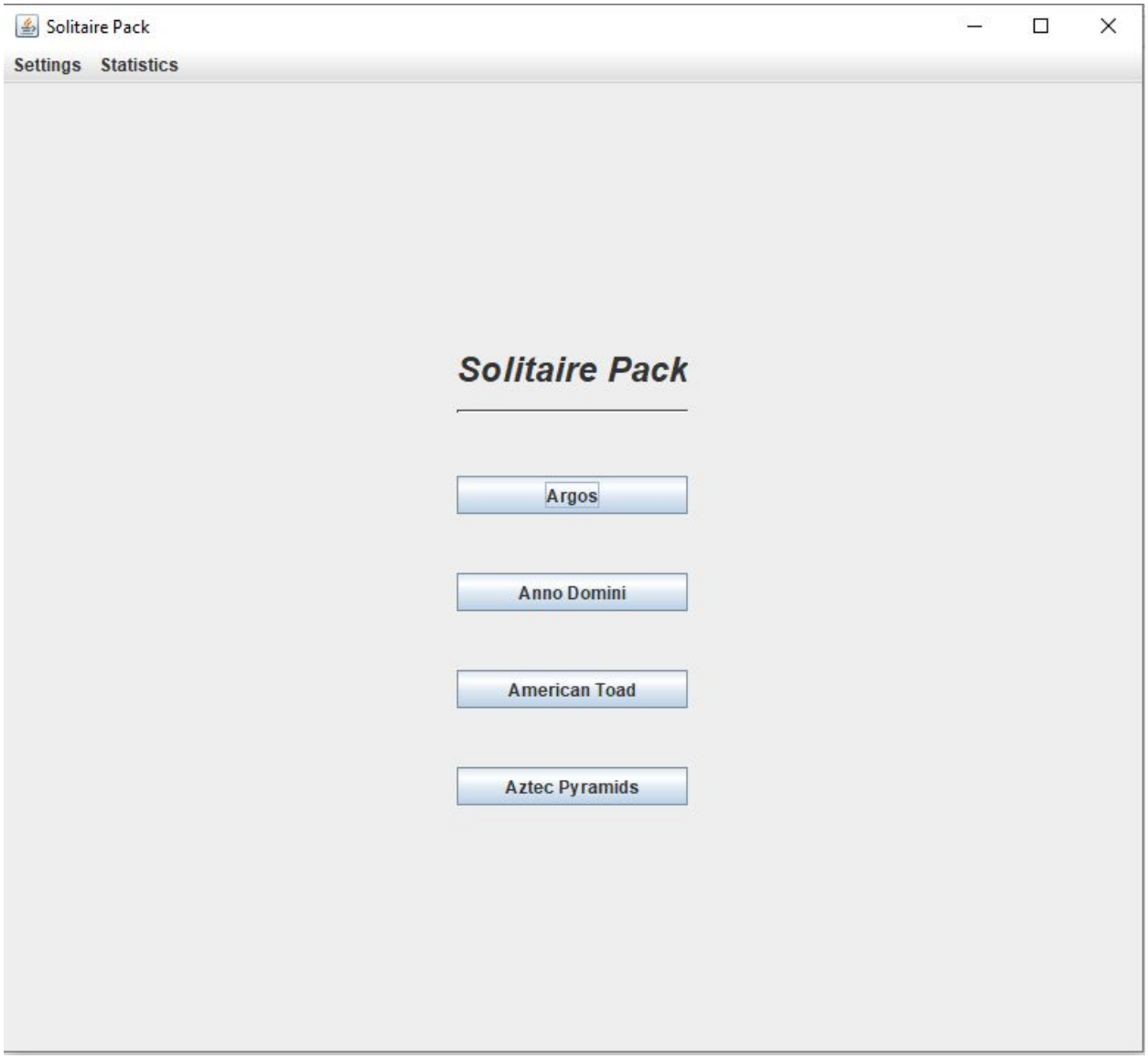
After unzipping the SolitarePackFinal, Your end result will be a folder.



After going into the SolitarePackFinal folder you should see some .pdf files and a runnable .jar.



You will want to double click the SolitairePackFinal.jar and run it as a .jar file. You're computer may ask you if you really want to run this and make sure you say yes to all security checks. After doing this you should have a running game with the main menu popping up as shown below.



Once you've reached this screen you're ready to move to the User's guide for a take on how to navigate our menus and Anno Domini!

6.2 USER'S GUIDE

After following the installation guide you should be ready to play any of the four games provided! The screen shown above shows our main menu. There are 6 unique things to take away from the main menu

1. Settings
    a. Our settings menu is a simple toggle on/off for every option. They are as follows.
        i. Show Argos - Show or hide Argos.
        ii. Show Anno Domini - Show or hide Anno Domini.
        iii. Show American Toad - Show or hide American Toad.
        iv. Show Aztec - Show or hide Aztec Pyramids.
        v. Volume toggle - Volume on/off.
2. Statistics
    a. Our statistics option has four choices, one for each game, They are as follows.

i. Argos - Opens statistics for Argos.
ii. Anno Domini - Opens statistics for Anno Domini.
iii. American Toad - Opens statistics for American Toad.
iv. Aztec Pyramids - Opens statistics for Aztec Pyramids.

b. After opening the statistics menu for the game you want an entirely new window opens. This new window has 7 different features on it. They are as follows.
i. Select
1. This menu lets you choose to open a statistics window for another game without closing the currently open statistics window you already have open.
ii. Title - Lists the title of the game currently selected.
iii. Total Games - Raw number of games played.
iv. Wins - The number of wins for the game currently selected.
v. Win Percentage - The win percentage of the game currently selected.
vi. Average Time - The average time taken playing the game currently selected.
vii. Best Time - The fastest time you've ever achieved winning on the given game.

c. Do note that all statistics are local to your machine and will stay stored on the machine even if you close out the Solitaire game overall.

3. Argos
a. Clicking the argos button allows you to play Argos!
4. Anno Domini
a. Clicking the Anno Domini button allows you to play Anno Domini!
5. American Toad
a. Clicking the American Toad button allows you to play American Toad!
6. Aztec Pyramids
a. Clicking the Aztec Pyramids button allows you to play Aztec Pyramids!

Now that we've looked at the features of the Main Menu lets take a closer look into the features of our given game Anno Domini. Do note that many of the features from our Anno Domini game will apply to the other games as well, however we ask that you refer to the User Guide of each individual game to get a deep understanding of all features that each Solitaire game provided has to offer.

Anno Domini:

1. Upon first opening Anno Domini you will notice 3 features on the bar at the top of your screen. They are as follows:
a. Select - Returns you to the main menu or any of the three alternate games! There is also a volume toggle checkbox here.
b. Rules - Displays the rules for Anno Domini.
i. After clicking the rules button a PDF will open up displaying the rules for playing Anno Domini. These will explain to you how to play Anno Domini as well as all the restrictions put on you as a player.
c. Volume Toggle - Volume on/off.
2. After reading up on the rules you are ready to play Anno Domini! Good luck and have fun.

6.3 ADMINISTRATOR GUIDE (POSSIBLY)

SECTION 7 - TEST AND VALIDATION

7.1 TEST OBJECTIVES

To be able to ensure that all the capabilities and functions of the game work and run as planned.

7.2 TEST ENVIRONMENT

We will be doing the testing in a local Java environment with a JDK version 11.0.

7.3 TEST CASES

7.3.1 MAIN MENU

Case ID: 01

Description: The buttons on the main menu for each game should pull up the selected game when clicked.

Expected result: When a game is selected, an instance of the game will appear for the user to play.

Actual result: All games are able to appear when selected in the main menu or select menu.

Pass/Fail: Pass

Case ID: 02

Description: The game shall have the ability to mute the game if the user wants the volume to be muted at any point, even during a game.

Expected result: When you tick the box next to *Volume Toggle*, the volume for the game will be muted.

Actual result: The volume stays muted when the box next to *Volume Toggle* is unticked and stays muted until the box is ticked again. The user has the ability to mute the game once they have started playing any of the games.

Pass/Fail: Pass

Case ID: 03

Description: The game shall have the ability to let the user select which games will appear on the main menu screen.

Expected result: When the box next to the game becomes unticked by the user, the button for the game will disappear from the main menu to remove clutter.

Actual result: When a game type is unchecked in the *Settings* menu, the game will disappear from the main menu. If the user decides later they want it, then they can tick the box next to the game and it will reappear.

Pass/Fail: Pass

Case ID: 04

Description: The user should be able to select the statistics for each game.

Expected result: When a user selects a game under the *Statistics* menu, a new window will appear that displays the statistics for that game.

Actual result: When you click a game under the *Statistics* menu, a new window pops up with the statistics for the given game chosen.

Pass/Fail: Pass

7.3.2 ANNO DOMINI GAME

Case ID: 05

Description: The rules must be accessible from within Anno Domini from any point within the game. Able to access the instructions at the beginning, four random points during the game, and at completion of the game.

Expected result: The instructions are accessible at any point during the play of the game, before it is started, and after it has finished.

Actual result: When the user selects *Rules* > *Open*, a browser window appears with a PDF that has the rules for Anno Domini.

Pass/Fail: Pass

Case ID: 06

Description: Each card or stack of cards functions as a clickable object within the game. In the case of stacks of cards that cannot be changed, those will provide no response. In the case of usable cards or stacks that can be changed, those functions are provided to match the game instructions.

Expected result: All usable cards and stacks function as they should. Cards and stacks that are off limits are impossible to manipulate as the user. The game instructions make it clear which cards and stacks are available for manipulation.

Actual result: The results for valid and invalid card placements were working in accordance with the rules of the game, *Anno Domini*. When a incorrect card was place, the card went back to its original place with a smooth animation.

Pass/Fail: Pass

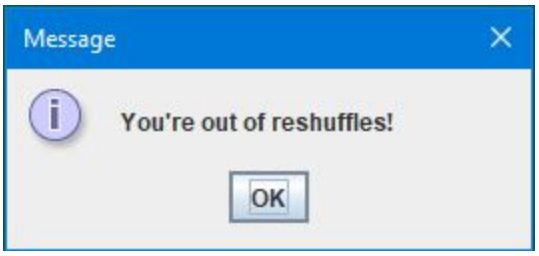
Case ID: 07

Description: The game must signal to the user when they have finished the game. This could be a message to signal they have succeeded or failed in their objective. In the case of Anno Domini, this means that the requirements for the correct year are met, the proper card stacks are complete, or that you have run out of options and failed to complete the game.

Expected result: Signal successfully sent to user at the time the game finishes. If the user has failed to complete the game, it gives a different message than it does at the time of successful completion.

Actual result: When a user has won the game, they will be presented with a winning animation and tell the player that they have won. If the user has lost the game, they will be told they are out of reshuffles.

Pass/Fail: Pass



7.4 VALIDATION OF USERS NEEDS

1. User requirement: The game shall be easy to be operate and navigate through.

Actual delivery: The menus for the game are very easy to navigate through

Pass/Fail: Pass

---

2. User requirement: The sounds for the game shall not be annoying and should be able to be silenced when desired.

Actual delivery: The game's select menu has a music toggle checkbox that turns the music on or off in-game.

Pass/Fail: Pass

---

3. User requirement: The game needs to be able to be run locally on a machine without access to the Internet.

Actual delivery: The game is able to run locally on a machine that has Java and the user has access to the source code or has the Jar file.

Pass/Fail: Pass

---

4. User requirement: The UI shall not be cluttered and full of advertisements, if they are a must then they should be tasteful and entertaining.

Actual delivery: There are no advertisements on the game.

Pass/Fail: Pass

---

5. User requirement: The application shall contain user instructions or online help in order to help the user know how to play the game.

Actual delivery: There is a rule set for *Anno Domini* when you select *Rules* > *Open* once the game as been started.

Pass/Fail: Pass

---

6. User requirement: The game must be able to keep track of statistics from past games from each user. The statistics may include, but not necessarily be limited to, the number of games played, games won, win percentage, and best time.

Actual delivery: The statistics page does show all of the following statistics.

Pass/Fail: Pass
